Edit on GitHub (https://github.com/liferay/liferay-docs/blob/6.2.x/develop/tutorials/articles/128-customizing-liferay-portal-with-hooks/06-overriding-a-portal-service-using-a-hook.markdown) (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/overriding-a-portal-service-using-a-hook?p_p_lifecycle=2&p_p_resource_id=kbArticleRSS&p_p_cacheability=cacheLevelFull&_2_WAR_knowledgebaseportlet_resourcePrimKey=523378&_2_WAR_knowledgebaseportlet_resourceClassNameId=10441)

OVERRIDING A PORTAL SERVICE USING A HOOK

Liferay Portal and its core portlets offer a host of services that you can programmatically invoke from local and remote clients. One only needs to look at Portal's service API (http://docs.liferay.com/portal/6.2/javadocs/com/liferay/portal/service/package-summary.html) or the APIs of its portlets, like the Document Library services API (http://docs.liferay.com/portal/6.2/javadocs/com/liferay/portlet/documentlibrary/service/package-summary.html), to see that Liferay is chock-full of useful access points. But you may encounter situations in which you want to modify the behavior of these services.

To do this, you may be tempted to extend a service interface directly. But there are problems inherent with this approach. Fix packs later added to the product may modify the interface (e.g., adding a new method to the service). If you've implemented the API directly, your implementation may not account for the modified interface. As a result, the patch could break your customization plugin. Don't worry–Liferay has provided a safe way to customize its services.

All the functionality provided by Liferay is enclosed in a layer of services that are accessed by the controller layer in its portlets; this architecture lets you change how a Liferay core portlet behaves without changing the portlet itself. Liferay generates dummy wrapper classes for all its service interfaces. For example, `UserLocalServiceWrapper` is created as a wrapper for `UserLocalService`, a service interface for adding, removing, and retrieving user accounts. If you extend the wrapper class, you can alter the service's behavior, and your customization can be safeguarded from being broken by any future patches to the interface.

This tutorial shows you how to modify a portal service using a hook. By the end of this tutorial, you'll have a hook plugin that overrides a Liferay service.

IMPLEMENTING THE PORTAL SERVICE OVERRIDE

Hook plugins are the best tool for leveraging this architecture to customize portal service behavior. To modify the functionality of a service from a hook,

/-/knowledge_base
/6-2/plugins-sdk)

Developing Plugins with Liferay IDE (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/liferay-ide)

Developing with Maven (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/maven)

Deploying Plugins (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/deploying-plugins)

MVC Portlets (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/developing-jsp-portlets-using-liferay-mvc)

JSF Portlets with Liferay Faces (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/jsf-portlets-with-liferay-faces)

Service Builder and Services (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/service-builder)

Security and Permissions (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/security-and-permissions)

you create a class that extends the service wrapper class, override the methods you want to modify, and instruct Liferay to use your service class to override those of the default class.

You can follow these steps to override any Liferay service from your own hook plugin:

1. Create a Liferay Hook plugin project in a Liferay Plugins SDK project (/develop/tutorials/-/knowledge_base/6-2/creating-a-hook-project-in-the-plugins-sdk) or Maven project (/develop/tutorials/-/knowledge_base/6-2/developing-liferay-hook-plugins-with-maven).

2. Create a class that extends the wrapper class of the service interface you want to override.

   To create the extension class from Liferay IDE/Developer Studio, open your project's `liferay-hook.xml` file, which is found in the `docroot/WEB-INF/` folder. Select the `liferay-hook.xml` file editor's *Overview* mode tab and select *Service Wrappers* from the editor's outline. Select *Add a Service Wrapper* in the editor's main area to bring up the Service Wrapper Detail options.

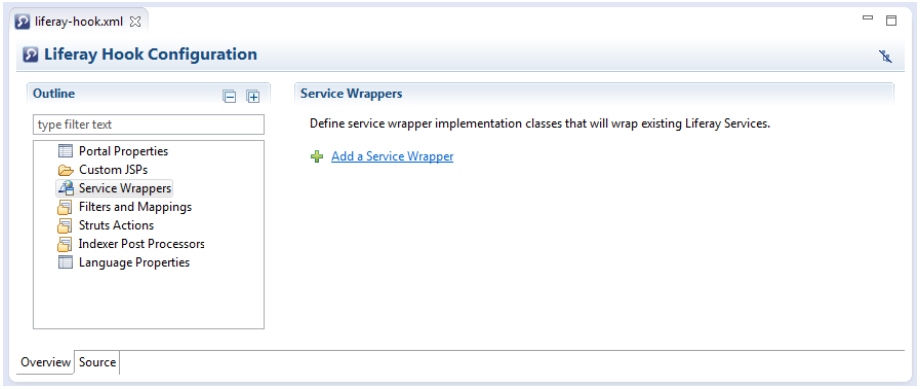

Figure 1: Liferay IDE's Hook Configuration editor comes with custom service wrapper creation and editing capabilities.

In the Service Wrapper Detail screen, click the `Browse` icon at the right of the Service Type text field and select the service class you want to override. In the Service Impl text field, you can enter the fully qualified class name of your service wrapper extension class and click the `Create` icon to the right of the text field. This creates the extension class you entered and opens it in Liferay IDE.

Search and Indexing (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/search-and-indexing)

Localization (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/localization)

Asset Framework (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/asset-framework)

Recycle Bin (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/recycle-bin)

Message Bus (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/using-liferays-message-bus)

Workflow (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/workflow)

JavaScript in Liferay (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/javascript-in-liferay)

User Interfaces with AlloyUI (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/alloyui)

User Interfaces with the Liferay UI Taglib

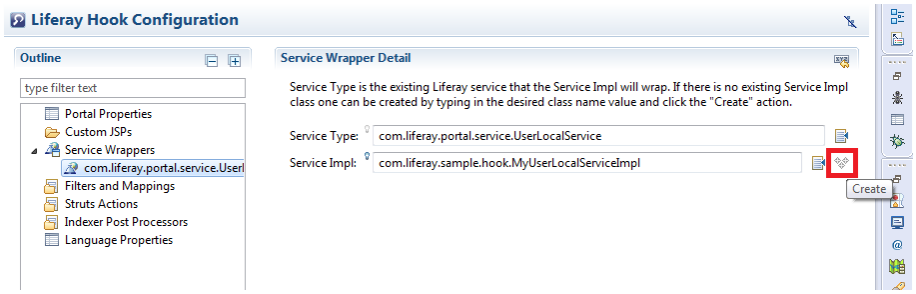

Figure 2: Creating wrapper extensions is easy. You enter the name of your service implementation class and click the *Create* icon to create it for overriding the service.

You can alternatively create your wrapper extension class manually in your favorite editor.

The initial wrapper extension class that Liferay IDE creates is virtually a blank canvas on which you can add your custom override methods. The wrapper class refers to Liferay's default implementation for the service. By calling the wrapper's methods via the `super.[methodName](...)` in your custom method implementations, you invoke the underlying default implementation.

The code below is from an example custom service implementation class named `MyUserLocalServiceImpl.java` that overrides the `UserLocalService` by extending `UserLocalServiceWrapper`. Note that its constructor calls the parent class constructor. A new implementation of the service's `deleteUser(...)` method has been added for holding custom logic. The parent class' version of the method, which invokes the underlying default implementation, is called at the end of the custom method. Calling the parent class method is optional but is a common practice to leverage Liferay's default implementation.

(https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/liferay-ui-taglibs)

Liferay Faces Alloy UI Components (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/liferay-faces-alloy-ui-components)

Liferay Faces Portal UI Components (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/liferay-faces-portal-ui-components)

Liferay Faces Bridge UI Components (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/liferay-faces-bridge-ui-components)

Android Apps with Liferay Screens (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/android-apps-with-liferay-screens)

iOS Apps with Liferay Screens (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/ios-apps-with-liferay-screens)

Mobile SDK (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/mobile)

Using the Device Recognition API

```
package com.liferay.sample.hook;

import com.liferay.portal.service.UserLocalService;
import com.liferay.portal.service.UserLocalServiceWrapper;

public class MyUserLocalServiceImpl extends UserLocalServiceWrappe
r {

    public MyUserLocalServiceImpl(UserLocalService userLocalServic
e) {
        super(userLocalService);
    }

    @Override
    public com.liferay.portal.model.User deleteUser(long userId)
        throws com.liferay.portal.kernel.exception.PortalException
,
            com.liferay.portal.kernel.exception.SystemException {

        // TODO Add your custom implementation of the method here

        // Optionally, you can call Liferay's default implementati
on via the wrapper

        return super.deleteUser(userId);
    }
}
```

Now that you've created your wrapper extension class, you can add methods to it to override Liferay's implementation of the service interface.

> **Note:** On deployment, the wrapper class extension is loaded in the hook's class loader, which means the extension can access any other class included in the same WAR file but *cannot* access Liferay's *internal* classes.

3. You must specify your custom service implementation class in the `liferay-hook.xml` file. On creating wrapper extension classes using Liferay IDE's Hook Configuration editor, Liferay IDE automatically specifies the service implementation class in the `liferay-hook.xml` file. If you create your wrapper extension class manually, you must also manually specify the service implementation class in a `<service></service>` element within the `<hook></hook>` element. See the `liferay-hook.xml` file's DTD (http://www.liferay.com/dtd/liferay-hook_6_2_0.dtd) for details.

(https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/using-the-device-recognition-api)

OpenSocial Gadgets (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/opensocial-gadgets)

Themes and Layout Templates (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/themes-and-layout-templates)

Application Display Templates (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/application-display-templates)

Customizing Liferay Portal with Hooks (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/customizing-liferay-portal)

Creating a Hook Project in the Plugins SDK (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/creating-a-hook-project-in-the-plugins-sdk)

Overriding Web Resources (https://dev.liferay.com/develop/tutorials/-/knowledge_base

For example, here's what a wrapper extension specification to `UserLocalService` can look like:

```
<hook>
    <service>
       <service-type>com.liferay.portal.service.UserLocalService</
service-type>
       <service-impl>com.liferay.sample.hook.MyUserLocalServiceImp
l</service-impl>
    </service>
</hook>
```

4. Deploy (/develop/tutorials/-/knowledge_base/6-2/deploying-plugins) your hook to your portal.

Your hook substitutes the service's default behavior with the behavior of your custom implementation.

There are other Liferay services that you may need to extend to meet advanced requirements. Here are just a few services that you may want to customize:

- `OrganizationLocalService` (http://docs.liferay.com/portal/6.2/javadocs/com/liferay/portal/service/OrganizationLocalService.html): Adds, deletes and retrieves organizations. Also assigns users to organizations and retrieves the list of organizations of a given user.
- `GroupLocalService` (http://docs.liferay.com/portal/6.2/javadocs/com/liferay/portal/service/GroupLocalService.html): Adds, deletes and retrieves sites.
- `LayoutLocalService` (http://docs.liferay.com/portal/6.2/javadocs/com/liferay/portal/service/LayoutLocalService.html): Adds, deletes, retrieves and manages pages of sites, organizations and users.

For a complete list of available services and their methods, check the Javadocs for Liferay Portal 6.2 Services (http://docs.liferay.com/portal/6.2/javadocs/com/liferay/portal/service/package-summary.html) or browse http://docs.liferay.com/portal/6.2/javadocs/ (http://docs.liferay.com/portal/6.2/javadocs/) for Javadocs on the services of any of Liferay Portal's core portlets. To access Javadocs for a different version of Liferay, visit http://docs.liferay.com/portal (http://docs.liferay.com/portal), select the Liferay Portal version, and click on the *Javadocs* link.

> **Note:** To modify a portal utility class, you can extend the utility's base implementation in a hook. But first, check Liferay's `portal.properties` (http://docs.liferay.com/portal/6.2/propertiesdoc/portal.properties.html) file to see if there's an option for specifying an extension to the utility. For example, to customize the behavior of

Liferay's `SanitizerUtil` (http://docs.liferay.com/portal/6.2/javadocs/com/liferay/portal/kernel/sanitizer/SanitizerUtil.html) class, you can extend the `BaseSanitizer` (http://docs.liferay.com/portal/6.2/javadocs-all/com/liferay/portal/kernel/sanitizer/BaseSanitizer.html) class with your custom implementation. Then you'd set the `sanitizer.impl` (http://docs.liferay.com/portal/6.2/propertiesdoc/portal.properties.html#Sanitizer) property to the fully qualified name of your implementation class.

You've done well learning how to properly customize Liferay services. Now get out there and put your newfound skills to use!

RELATED TOPICS

Developing Plugins with the Plugins SDK (/develop/tutorials/-/knowledge_base/6-2/plugins-sdk)

Developing Liferay Hook Plugins with Maven (/develop/tutorials/-/knowledge_base/6-2/developing-liferay-hook-plugins-with-maven)

Application Display Templates (/develop/tutorials/-/knowledge_base/6-2/application-display-templates)

0 (0 Votes)

Customizing Sites and Site Te... (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/customizing-sites-and-site-templates-with-application-adapters)

Performing a Custom Action U... (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/performing-a-custom-action-using-a-hook)

(https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/performing-a-custom-action-using-a-hook)

Creating Model Listeners (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/creating-model-listeners)

Overriding and Adding Struts Actions (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/overriding-and-adding-struts-actions)

Extending the Indexer Post Processor Using a Hook (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/extending-the-indexer-post-processor-using-a-hook)

Supporting Right-to-Left Languages in Plugins (https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/supporting-right-to-left-languages-in-plugins)

Creating Plugins to Extend Plugins (https://dev.liferay.com/develop/tutorials

/-/knowledge_base
/6-2/creating-
plugins-to-extend-
plugins)

Advanced
Customization
with Ext Plugins
(https://dev.liferay.com
/develop/tutorials
/-/knowledge_base
/6-2/advanced-
customization-
with-ext-plugins)

Audience Targeting
(https://dev.liferay.com
/develop/tutorials
/-/knowledge_base
/6-2/audience-
targeting)

Importing Resources
(https://dev.liferay.com
/develop/tutorials
/-/knowledge_base
/6-2/leveraging-
the-resources-
importer)

Modularization with
OSGi Plugins
(https://dev.liferay.com
/develop/tutorials
/-/knowledge_base
/6-2/developing-
osgi-plugins-
for-liferay)

Plugin Security and
PACL
(https://dev.liferay.com
/develop/tutorials
/-/knowledge_base
/6-2/plugin-security-
and-pacl)

DOWNLOADS

Portal (http://www.liferay.com/downloads/liferay-portal/available-releases)

Social Office (http://www.liferay.com/downloads/social-office/available-releases)

Sync (http://www.liferay.com/downloads/liferay-sync)

Liferay Faces (http://www.liferay.com/community/liferay-projects/liferay-faces/download)

OTHER LIFERAY SITES

(http://www.liferay.com) (http://alloyui.com) (http://issues.liferay.com

PRIVACY POLICY (HTTPS://WWW.LIFERAY.COM/ABOUT-US/PRIVACY)

/

MEET THE TEAM (/MEET-THE-TEAM)

© 2014 LIFERAY ALL RIGHTS RESERVED.